UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/700,338	11/03/2003	Lewis K. Cirne	WILY-01013US0	5180

28554 7590 03/05/2009
Vierra Magen Marcus & DeNiro LLP
575 Market Street, Suite 2500
San Francisco, CA 94105

EXAMINER	
WEI, ZHENG	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
03/05/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

	Application No.	Applicant(s)
Office Action Summary	10/700,338	CIRNE ET AL.
	Examiner	Art Unit
	ZHENG WEI	2192

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *13 November 2008*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-3,5-15,17-24,26-30,32-35,37-42,44-49 and 51-54* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-3, 5-15, 17-24, 26-30, 32-35, 37-42, 44-49 and 51-54* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

Remarks

1. This office action is in response to the amendment filed on 11/13/2008.
2. Claims 2, 3, 5-12, 14-15, 17-21,23-24, 26-30, 32, 34-35, 37-38, 41-42, 44-49, and 51-52 have been amended.
3. Claims 31 and 50 have been cancelled
4. Claims 53 and 54 have been added.
5. The objection to claims 2, 3, 5-12, 14, 15, 17-21, 23, 24, 26-32, 34, 35, 37-38, 41-42 and 44-52 has been cancelled in view of Applicants amendment and clarification.
6. Claims 1-3, 5-15, 17-24, 26-30, 32-35, 37-42 and 44-49, and 51-54 remain pending and have been examined.

Response to Arguments

7. Applicant's arguments filed on 11/13/2008, in particular on pages 13-19, has been fully considered.
   - At page 13, third paragraph, Applicants submit that neither Berkley nor Webster, alone or in combination , disclose "said step of determining whether to modify said method includes determining whether said method calls another method" as claimed. Moreover, the Applicants assert that previous Office Action is improperly relying upon Official Notice at last paragraph of page 14. However, Examiner's position is that Berkley discloses inserting a

trace function into an existing application according to the configuration settings. Webster discloses a method to parse and find each of the methods in the bytecode (class) (see for example, Fig.3, step 32, “CL Loads Classes”, step 325 “Is Class Method in DG?”). Therefore, as Webster finding the class method by parsing or examining each line of instructions in the bytecode, it would have been obvious to one having ordinary skill in the art at the time the invention was made to find/identify access level keywords e.g. public or package or private; compilation flag for synthetic and method calling other methods by just simply examining the parsed bytecode instructions in Webster. Because access keywords, compilation flag and calling/called methods are unique identified by the keywords, e.g. public, private or package. However, for the purpose of clarification, a new reference is incorporated below which uses call graph to indicate the method calling other methods.

Specification

8. The disclosure is objected to because of the following informalities: term “non-synthetic” recited in the claims 2, 5, 14, 17, 23, 26, 34, 37, 41, 47, 48 and 52 have not been defined in the specification. See for M.P.E.P. 608.01(o). Appropriate correction is required.

Claim Objections

9. Claim 32 is objected to because the numbering of claims is not in accordance with 37 CFR 1.126 which requires the original numbering of the claims to be preserved throughout the prosecution. When claims are canceled, the remaining claims must not be renumbered. When new claims are presented, they must be numbered consecutively beginning with the number next following the highest numbered claims previously presented (whether entered or not). For the purpose of compact prosecution, the misnumbered claim 32 has been renumbered to depend on claim 22.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

10. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

11. Claims 2, 3, 5, 14, 15, 17, 23, 24, 26, 34, 37, 41, 42, 47, 48 and 52 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Said claims recite terms “non-synthetic”, “access level of public or package” respectively. The terms are relative terms which render the claim indefinite. The term “non-synthetic” and “access level of public or package” are not defined by the claims, the specification does not provide a standard for ascertaining the requisite degree, and one of ordinary skill in the art would not be

reasonably apprised of the scope of the invention. For the purpose of compact prosecution, the Examiner treats "non-synthetic" as methods other than synthetic method in Java programming language and treats "access level of public or package" as the special terms defined in Java programming language.

Claim Rejections - 35 USC § 103

12. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

13. Claims 1-3, 5-8, 10, 12-15, 17, 18, 20, 39-42, 44 and 47-54 are rejected under 35 U.S.C. 103(a) as being unpatentable over Berkley (Berkley et al., US 6351843 B1) in view of Webster (US6,738,965 now cited as prior art in this office action) in further view of Grove (Grove et al., Call Graph Construction in Object-Oriented Languages, now is recorded as prior reference)

Claim 1:

Berkley discloses a process for monitoring, comprising:

- accessing a method (see for example Fig.5, step 350 and related text, also see, col.2, lines 36-37, "Further, the system includes means for running the application executable using the modified runtime configuration settings");
- determining whether to modify said method, said step of determining whether to modify said method (see for example, Fig.1 and related text, "METHOD B",

"METHOD C" and also see col.2, lines 39-40, "means for determining whether the function (method) is active for a class of the executable using the modified configuration settings"); and

- modifying said method for a particular purpose if said method calls another method. (see for example, col.2, lines 41-43, "means for dynamically creating a redirection stub to insert the function for the class if the function is active for that class")

<u>Berkley</u> also discloses a method to provide configuration settings to specify user interested information/methods that need to be traced and further determining whether to modify said method to insert trace function according the user's settings (see for example, Fig.5, item 300 "Configuration Settings" item 310 "Add Setting to Specify trace for Desired Class:, item 320 "New Configuration Settings and related text). But <u>Berkley</u> does not explicitly disclose the determination includes determining whether said method calls another method. However, <u>Webster</u> in the same analogous art of selective tracing methods discloses the similar solution that a user can specify particular trace information of interest and provides a selection of methods to be traced from a program (see for example, Summary and Fig.3 item 325 and related text). Grove discloses a method to construct a call graph (see for example, Fig.1 and related text; also see p.109, section 2.1, first paragraph). As Webster disclosed that different users have different purpose/interest to trace/debug different methods (see for example, ABSRACT, "selection of method to be traced"), it is obvious said method also

including those methods calling other methods as indicated by Grove's call graph. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to trace/debug user interested methods by identify/determining the method calling other method using Webster and Grove's method and further configuring and inserting tracing function using Berley's method. One would have been motivated to do so to specify and trace particular trace information of interest as suggested by <u>Webster </u>(see for example, col.2, lines 34-35, "A user can specify particular trace information of interest...")

Claims 2 and 14:

<u>Berkley</u> discloses processes according to claims 1 and 13 above respectively, but does not explicitly disclose said step of determining whether to modify said method includes determining whether said method is non-synthetic. However, It is well known in the Java programming that all synthetic methods generated by Java compiler are flagged in the class file and thus are easily identified (A well known and widely used Java programming standard: Java Virtual Machine Specification). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method is non-synthetic by checking the synthetic attribute field in bytecode while being compiled by JIT, Hotspot runtime or other bytecode scanning tools. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by <u>Berkley</u>

(see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claims 3 and 15:

Berkley discloses processes according to claim 1 and 13 above respectively, but does not explicitly disclose said step of determining whether to modify aid method includes determining whether said method has an access level of public or package. However, it is well known in the Java programming that JVM specification (see for example, a well known and widely used Java Virtual Machine Specification) defines a set of access flags in method_info structure which has a flag name "ACC_PUBLIC" for access level of public or package. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method has an access level of public or package by using JIT, Hotspot runtime or other bytecode scanning tools to check this flag to determining whether said method has an access level of public or package. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claims 5 and 17:

Berkley discloses processes according to claim 1 and 13 above respectively, but does not disclose said step of determining whether to modify said method includes determining whether said method is non-synthetic, calls another method and has an access level of public or package. However, according to the rejection for the claims 2, 3, 4 and 14, 15 above, it would have been obvious to one having ordinary skill in the art at the time the invention was made to combine those steps together to further focus on tracing specify one or more methods for which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claims 6 and 18:

Berkley discloses processes according to claim 1 and 13 above respectively, but does not disclose said step of determining whether to modify said method includes determining whether said method calls one or more different methods and can be called by a sufficient scope of one or more other methods. However, it is well known in the Java programming that JVM specification (see for example, a well known and widely used Java Virtual Machine Specification) defines method by using a block starting with the tag "Method" that contains the information about calling other methods in java bytecode. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method calls another method and can be called by a sufficient scope of one or more other methods by checking the method information in that block while running by JIT in JVM or other bytecode scanning tools. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 7:

Berkley further discloses a process according to claim 1, wherein: said step of modifying includes modifying object code (see for example, col.2, lines 45-46, "inserting a function into an application executable without recompiling the executable.")

Claim 8:

Berkley also discloses a process according to claim 1, wherein: said step of modifying includes adding a tracer for said method (see for example, Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…").

Claim 10:

Berkley further discloses a process according to claim 1, wherein: said step of modifying includes adding exit code and start code to existing object code (see for example, Fig.6, step 460 and related text, "Create redirection stubs that will call trace entry and ext method around target method").

Claim 12:

Berkley also discloses a process according to claim 1, wherein: said particular purpose is to add a first tracer (see for example, Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…").

Claim 13:

Claim 13 is another version process for monitoring as in claim 1 addressed above, wherein all claimed limitation functions have been addressed and/or set forth above. Therefore, it also would have been obvious.

Claim 20:

Berkley further discloses a process according to claim 13, wherein: said step of using a first tracing mechanism includes modifying existing object code to add said first tracing mechanism (see for example, Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…").

Claim 39:

Berkley discloses an apparatus capable of monitoring, comprising:

- means for determining whether a method is call another method (see for example, Fig.1 and related text, "METHOD B", "METHOD C" and also see col.2, lines 39-40, "means for determining whether the function (method) is active for a class of the executable using the modified configuration settings"); and

means for tracing said method for a particular purpose only if said method calls another method (see for example, col.2, lines 41-43, "means for dynamically creating a redirection stub to insert the function for the class if the function is active for that class", also see Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…").

Claim 40:

Berkley discloses an apparatus capable of monitoring, comprising:

- a storage device (see for example, Fig.3, items 102, 103 "Main Storage", "External Storage Media" and related text); and

- one or more processors in communication with said storage device (see for example, Fig.3, item 104, "CPU 1…CPU N" and related text), said one or more processors perform a process comprising:
- accessing a method (see for example Fig.5, step 350 and related text, also see, col.2, lines 36-37, "Further, the system includes means for running the application executable using the modified runtime configuration settings");
- tracing said method for a particular purpose if said method calls one or more different methods and can be called by a sufficient scope of one or more other methods (see for example, col.2, lines 41-43, "means for dynamically creating a redirection stub to insert the function for the class if the function is active for that class", also see Fig.5, step 360 and related text, also see, col.3, lines 6-8, "To restate, a technique is presented for dynamically modifying class lineage in order to insert a function, such as a trace function…").

<u>But Berkley does not disclose</u>:

- determining whether said method calls one or more different methods and can be called by a sufficient scope of one or more other methods.

However, it is well known in the Java programming that JVM (Java Virtual Machine) specification defines method by using a block starting with the tag "Method" that contains the information about calling other methods in java bytecode. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method calls another method and can be called by a sufficient scope of one or more other methods by checking the method information in that block while running by JIT in JVM or other bytecode scanning tools. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 41:

Berkley discloses an apparatus according to claim 40, but does not explicitly disclose said step of determining whether to modify said method includes determining whether said method is non-synthetic. However, it is well known in the Java programming that all synthetic methods generated by Java compiler are flagged in the class file and thus are easily identified (see for example, a well known and widely used Java Virtual Machine Specification). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method is non-synthetic by checking the synthetic attribute field in bytecode while being compiled by JIT, Hotspot runtime or other bytecode scanning tools. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 42:

Berkley further discloses an apparatus according to claim 40, but does not explicitly disclose said step of determining whether to modify aid method includes determining whether said method has an access level of public or package. However, it is well known in the Java programming that JVM specification (see for example, a well known and widely used Java Virtual Machine Specification) defines a set of access flags in method_info structure which has a flag name "ACC_PUBLIC" for access level of public or package. Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to determine whether said method has an access level of public or package by using JIT, Hotspot runtime or other bytecode scanning tools to check this flag to determining whether said method has an access level of public or package. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 44:

Berkley discloses an apparatus according to claim 40 and further discloses said process further includes modifying existing object code for said method in order to add a first tracing mechanism (see for example, col.2, lines 45-46, "inserting a function into an application executable without recompiling the executable.")

Claims 47-54:

Claims 47-54 are another version process for monitoring as in claims 1-3, 5 and 8 addressed above, wherein all claimed limitation functions have been addressed and/or set forth above. Therefore, they also would have been obvious.

14. Claims 9, 11, 19, 21-24, 26-35, 37, 38, 45 and 46 are rejected under 35 U.S.C. 103(a) as being unpatentable over Berkley (Berkley et al., US 6,351,843) in view of Webster (US6,738,965) in further view of Berry (Berkley et al., US 6,662,359).

Claim 9:

Berkley and Webster disclose a process according to claim 1, but does not explicitly disclose said step of modifying includes adding a timer for said method. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, "timestamp" and related text"). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 11:

Berkley discloses a process according to claim 10, wherein:

- said start code starts a tracing process (see for example, Fig.6, step 460 and related text, "Create redirection stubs that will call trace entry and ext method around target method");
- said exit code stops said tracing process (see for example, Fig.6, step 460 and related text, "Create redirection stubs that will call trace entry and ext method around target method");
- said exit code is positioned to be executed subsequent to original object code (see for example, Fig.6, step 470 and related text, "Remaining class construction flows");

But Berkley does not disclose said steps of adding exit code including jump instruction, exception table and said step of adding an entry in said exception table. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses:

- said step of adding exit code includes adding an instruction to jump to said exit code from said original object code (see for example. Fig.8 steps 812-816 and related text, also see col.9, line 55- col.10, line 8,"a jump around inserted code");
- said step of adding exit code includes adding an entry in an exception table; and (see for example. Fig.8 step 802 and related text "Modify the exception table");
- said step of adding an entry in said exceptions table includes adding a new entry into said exceptions table for said method, said new entry indicates a range of indices corresponding to said original object code, said new entry includes a reference to said exit code and said new entry indicates that said new entry pertains to all types of exceptions (see for example. Fig.8 steps 812-816 and related text, also see col.9, line 55- col.10, line 8,"a jump around inserted code");

Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to add jump instruction and maintain exception table for inserting function into an application executable at runtime. One Would have been motivated to integrated Berry's steps into Berkley's process to ensure that code which moved due to either insertions or deletions is correctly relocated and related references are adjusted as pointed out by Berry (See for example, Col.9, lines 55-58, "to ensure that code which is moved due to either insertions or deletions is correctly relocated and related references are adjusted")

Claim 19:

Berkley discloses a process according to claim 13, but does not explicitly disclose said step of modifying includes adding a timer for said method. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, "timestamp" and related text"). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented")

Claim 21:

Berkley discloses a process according to claim 20, but does not explicitly disclose said step of modifying includes adding a timer for said method. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, “timestamp” and related text”). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, “dynamically inserting a function into an existing application executable”, “allows a user to specify one or more methods for which the function is to be implemented”)

Claims 22-24 and 26-32:

Claims 22-24 and 26-32 claim one or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, which is the product version of the process claims as discussed in claims 1-3 and 5-11 above respectively. Therefore, these claims are obvious over Berkley and Berry, because it is well known in the computer art to practice and/or produce such a program product for carrying out the acts/steps of such process by a typical computer processor.

Claims 33-35, 37 and 38:

Claims 33-35, 37 and 38 claim one or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, said processor readable code for programming one or more processors to perform a process as discussed in claims 13-15, 17 and 19 above respectively. Therefore, these claims are obvious over Berkley and Berry, because it is well known in the computer art to practice and/or produce such a program product for carrying out the acts/steps of such process by a typical computer processor.

Claim 45:

Berkley discloses an apparatus according to claim 44 above, but does not disclose said first tracing mechanism includes a timer. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, “timestamp” and related text”). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, “dynamically inserting a function into an existing application executable”, “allows a user to specify one or more methods for which the function is to be implemented”).

Claim 46:

Berkley discloses an apparatus according to claim 44 above, but does not disclose said step of tracing includes timing said method. However, Berry in the same analogous art of system and method for injecting hooks into java classes to handle exception and finalization processing discloses using timestamp (see for example, col.14, lines 1-19, column 3 in the example table, "timestamp" and related text"). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use timestamp as a way to trace specified application executable. One would have been motivated to do so to allow a user to trace specified one or more methods of which the function is to be implemented as suggested by Berkley (see for example, col.2, lines 7-14, "dynamically inserting a function into an existing application executable", "allows a user to specify one or more methods for which the function is to be implemented").

Conclusion

15. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

16. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Zheng Wei whose telephone number is (571)

270-1059 and Fax number is (571) 270-2059. The examiner can normally be reached on Monday-Thursday 8:00-15:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature of relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571- 272-1000.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Z. W./
Examiner, Art Unit 2192

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192